

Tree Search and Quantum Computation

Luís Tarrataca^{1*} and Andreas Wichert¹

GAIPS/INESC-ID

Department of Informatics

IST - Technical University of Lisbon - Portugal

{luis.tarrataca, andreas.wichert}@ist.utl.pt

Abstract. Traditional tree search algorithms supply a blueprint for modeling problem solving behaviour. A diverse spectrum of problems can be formulated in terms of tree search. Quantum computation, in particular Grover’s algorithm, has aroused a great deal of interest since it allows for a quadratic speedup to be obtained in search procedures. In this work we consider the impact of incorporating classical search concepts alongside Grover’s algorithm into a hybrid quantum search system. Some of the crucial points examined include: (1) the reverberations of contemplating the use of non-constant branching factors; (2) determining the consequences of incorporating an heuristic perspective into a quantum tree search model.

Keywords: quantum computation, tree search, heuristic

1 Introduction

The concepts of knowledge representation and the reasoning processes that support knowledge application have long been a key interest area in the field of artificial intelligence. Knowledge enables problem-solving agents to determine an appropriate action in order to better deal with complex environments. Reasoning allows an agent to perform complex decisions by employing a finite amount of knowledge [Luger and Stubblefield, 1993]. One such form of reasoning consists of classical tree searching. Tree search is employed whenever decisions must be made that are based on complex knowledge. Tree search algorithms play a crucial role in many applications, e.g. production systems [Post, 1943] [Newell et al., 1959] [Newell, 1963] [Ernst and Newell, 1969] [Anderson, 1983] [Laird et al., 1986] [Laird et al., 1987], game playing programs [Feldmann, 1993] [Hsu, 1999] [Hsu, 2002] [Campbell et al., 2002] and robot control systems [Santos et al., 2009b] [Santos et al., 2009a].

The next few section are organized as follows: Section 1.1 reviews some of the major tree search algorithms; Section 1.2 present an alternative search method based on quantum computation; Section 1.3 proposed an hybrid search system combining classical tree search algorithms alongside quantum search. Section 1.4 presents the objectives and associated problems.

1.1 Classical tree search

The simplest tree search algorithms rely on a “brute-force” approach. These methods perform an exhaustive examination of all possible sequences of moves until goal states are reached. The

* Luís Tarrataca was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and FCT grant DFRH - SFRH/BD/61846/2009.

search through the state space systematically checks if the current state is a goal state. If a non-goal state is discovered then the current state is expanded by applying a successor function, generating a new set of states. The choice of which state to expand is determined by a search strategy. In a great deal of occasions, an artificial intelligence application does not possess an adequate level of knowledge enabling the choice of the most promising state. Strategies that can only distinguish between goal states and non-goal states, without being able to determine if one state is more promising than another, are referred to as uninformed search strategies. Examples of uninformed search strategies include the well known breadth first search [Moore, 1959], depth-first search [Hopcroft and Tarjan, 1973], and also the iterative deepening search [Slate and Atkin, 1977]. Uninformed strategies are only successful for small problem instances. Typically, most problems search space is characterized by an exponential growth [Garey and Johnson, 1979]. Due to the mammoth dimensions of the search space it becomes impractical, both time- and space-wise, to perform an exhaustive examination.

Alternatively, it is possible to employ additional insights, that arise beyond the definition of the problem. The use of this information, thus the term informed search strategies, allows for solutions to be found more efficiently. Typically, informed search strategies employ an evaluation function $f(n)$ which considers a cost function $g(n)$ alongside a heuristic function $h(n)$. Function $g(n)$ can be interpreted as representing the cost to reach node n whilst $h(n)$ represents an estimate on the cost to reach a leaf node from node n . Traditionally, the node with the lowest evaluation value is selected for expansion. Examples of some of the best known informed search strategies include greedy search [Newell and G, 1965] and A* search [Hart et al., 1968]. More recent advances on informed strategies include IDA* [Korf, 1985] and RBFS [Korf, 1991], [Korf, 1993]. A time complexity comparative assessment between the various algorithms is presented in Table 1.

Search	Reference	Strategy	Time
Breadth-first	[Moore, 1959]	Uninformed	$O(b^{d+1})$
Depth-first	[Hopcroft and Tarjan, 1973]	Uninformed	$O(b^m)$
Iterative-deepening	[Slate and Atkin, 1977]	Uninformed	$O(b^d)$
Greedy	[Newell and G, 1965]	Informed	$O(b^m)$
A*	[Hart et al., 1968]	Informed	$O(b^d)$
IDA*	[Korf, 1985]	Informed	$O(b^d)$
RBFS	[Korf, 1991]	Informed	$O(b^d)$

Table 1: Tree Search Algorithm Comparison (b - branching factor, d - depth of a solution, m - maximum depth).

1.2 Quantum Search

Grover's algorithm performs a generic search for a solution using the principles of quantum computation and mechanics [Grover, 1996] [Grover, 1998a]. Suppose we wish to search through a problem's search space of dimension N . Also, consider that we are also capable of efficiently perceiving a solution to our problem. This is similar to the NP class of problems whose solutions are verifiable in polynomial time $O(n^k)$ for some constant k , where n is the size of the input to the problem [Edmonds, 1965]. Grover's search algorithm employs quantum superposition and reversible computation in order to query many elements of the search space simultaneously.

Grover's algorithm was later experimentally demonstrated in [Chuang et al., 1998]. The algorithm provides a polynomial speed-up when compared with the best-performing classical search algorithms. As previously mentioned, any such classical algorithm requires $O(N)$ time in order to search N elements. Grover's algorithm requires $O(\sqrt{N})$ time, providing a quadratic speedup, which is considerable when N is large.

Oracle unitary operator In quantum computation mathematical objects known as unitary operators are responsible for the time-evolution of the state of a close quantum system. This is one of the foundational principles behind quantum computation, and is also known as the evolution postulate [Kaye et al., 2007]. A black box, also referred to as an oracle [Nielsen and Chuang, 2000], representing a unitary operator U , is employed in order to indicate, through a reverse of the associated amplitude, which of the values present in an amplitude register corresponds to the searched ones. This process can be performed by adding an additional input bit c to the original n -bit input register x and performing a XOR operation. This behaviour is illustrated in Expression 1 which employs the ket notation introduced by Paul Dirac [Dirac, 1939] [Dirac, 1981]. Classical reversible circuitry can also be described through such a formulation, albeit without employing the ket notation.

$$U : |x\rangle|c\rangle \mapsto |x\rangle|c \oplus g(x)\rangle \quad (1)$$

Grover's algorithm employs a process of amplitude amplification, known as Grover's iterate, in order to amplify the amplitudes of the solutions and in the process diminish those of the non-solutions. This process is performed by setting the control register c to a specified eigenvector, which, when combined with Grover's iterate can be mathematically proven to perform an inversion about the mean of the amplitudes [Kaye et al., 2007]. As a direct result of Grover's iterate, the probability of an answer bearing state increases. However, the amplitude of the solution value is amplified only in a linear way. If the function f is provided as a black box, then $O(\sqrt{N})$ applications of the black box are necessary in order to solve the search problem with high probability for any input [Nielsen and Chuang, 2000]. *

Traditionally, classical computation is seen as an irreversible process, a direct consequence from the use of many-to-one binary gates. A logical gate is a function $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$ from some fixed number k of input bits to some fixed number l of outputs bits [Mano and Kime, 2002]. A computation is said to be reversible if given the outputs we can recover the inputs [Toffoli, 1980a] [Toffoli, 1980b]. Mathematically, a reversible computation corresponds to the concept of a bijective function. It turns out that there is a general mechanism for converting irreversible computations into reversible ones. Each irreversible gate can be made reversible by adding some additional input and output wires [Kaye et al., 2007]. This conversion introduces a certain number of inputs and outputs to each irreversible gate. It is this additional information that provides for reversible computation.

Intuitively, it should come as no surprise that an irreversible circuit can be made reversible by substituting each irreversible gate by an equivalent reversible gate [Toffoli, 1980a]. By

* A number of improvements have been proposed since Grover's original work [Grover, 2002] [Grover, 2005]. These improvements essentially targeted reduced time complexity bounds for non-query operations and overall robustness. For a number of several novel search related applications please refer to [Grover, 1998a] [Grover, 1998b] [Grover, 1999].

substituting each element of the circuit with its respective inverse we are able to perform the inverse operation of the original circuit. In practice, this means that if we run the reversed circuit with an output, we will obtain the originating input bit register.

Emil Post's research into the principals of mathematical logic and description of the complete sets of truth functions [Post, 1941] implied that all functions $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$ could be computed by binary circuits employing logical gates \neg , \vee and \wedge . Since it is always possible to build a reversible version of an irreversible circuit, reversible computation can also compute all functions $f : \{0, 1\}^k \rightarrow \{0, 1\}^l$.

Quantum superpositions In order to gain a “quantum advantage”, Grover assembles a quantum superposition containing all possible values that should be presented as input to the unitary operator. At its core, the superposition principle simply conveys the notion that multiple quantum states exist at the same time. Let $|\psi\rangle$ denote the n -bit input register presented to unitary operator U , then $|\psi\rangle$ takes the form illustrated in Expression 2.

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \quad (2)$$

Since unitary operators obey linearity principles we are now in a position to apply unitary operator U to the superposition register. In practice this process means that all values present in the superposition register are processed simultaneously. This operation is illustrated in Expression 3. Additionally, unitary operators as well as input registers can be described by matrices. Accordingly, in light of Expression 3, applying unitary operator U to input register $|\psi\rangle$, can be understood as performing matrix multiplication $U|\psi\rangle$.

$$U|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} U|x\rangle \quad (3)$$

1.3 Hybrid system

As previously mentioned, Grover's algorithm performs a generic search by employing a unitary operator U . It was designed with the purpose of searching an unstructured collection of registers. I.e. the algorithm's purpose can be understood as looking for binary strings within a collection. Albeit, since binary strings can be used to encode mathematical abstractions, can we build upon this behaviour in order to develop a hybrid system performing classical hierarchical search using Grover's algorithm? This hybrid approach would combine traditional tree search mechanisms with the efficiency gains promised by Grover's algorithm.

Assume that the unitary operator U employed during Grover's iterate is developed in such a way as to recognize potential solutions. Intuitively, it is fairly easy to see that U will need to contemplate the sequence of steps taken in each path of a tree in order to determine if it leads to a solution. However, it needs to do so by restricting itself to a binary representation. This coding strategy serves two direct purposes, namely (i) provide a numerical basis on which the development of U can be built-upon; (ii) being able to translate Grover's output into a set

of actions leading to a solution. These actions are to be interpreted as the set of decisions executed at each step of the tree search, leading from the root node to a goal state.

In order to formally introduce the coding mechanism lets start by considering the binary tree presented in Figure 1. The illustrated binary tree has a root node A . Also, has it is possible to see each layer of depth d provides an additional 2^d nodes to the tree. At each node it is always possible to apply two actions, i.e. we have a branching factor $b = 2$. Each possible path leading to a leaf node can thus be perceived as a concatenation of the binary strings encoding the associated set of performed actions. Using this approach we are able to build a superposition $|\psi\rangle$ containing all the possible paths. Superposition $|\psi\rangle$ and a unitary operator U can then be employed by Grover's algorithm to determine the paths leading to solutions.

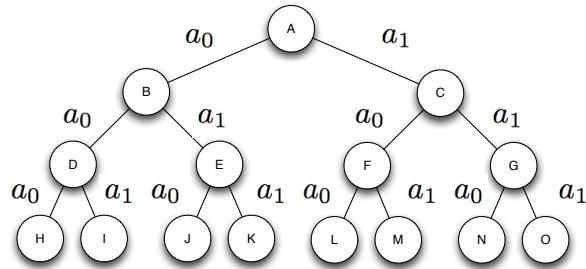


Fig. 1: A search tree with a constant branching factor $b = 2$, depth $d = 3$ for a total of $b^d = 8$ leaf nodes.

1.4 Objectives and Problems

As previously mentioned, unitary operators can be derived by transforming irreversible circuits into reversible ones. Accordingly, it is always possible to build a unitary operator U which checks if a set of actions produces a solution. In this work we will not be concerned with the actual implementation details of U but rather on how such a problem could potentially be approached. Additionally, we will also be interested in determining how such an hybrid system would be affected by traditional search concepts. For instance, what are the ramifications of a variable such as the branching factor? Would the system require the use of a constant branching factor? Clearly, this is not always the case for the complete set of problems that can potentially be addressed by search algorithms. When considering a non-constant branching factor, what would be the associated impacts in overall system performance? Additionally, traditional search strategies typically employ some kind of information to determine which states are more promising than others when trying to reach a goal state. Can an heuristic bounded procedure be incorporated into such an approach? If so, do we stand to gain any significant advantage?

These and other questions will be addressed in the remaining sections of this work. In Section 2 we will focus on assessing how such an hybrid system would perform from a branching factor perspective. In Section 3 we devote our efforts to researching on the impacts of incorporating

heuristic concepts into the hybrid proposal. Section 4 presents and discusses the parallels, alongside the differences, between our proposal and another well-known kind of graph inspection tool, respectively the quantum random walk. We present the conclusions of this work in Section 5. With these sections we hope to lend some intuition into the advantages, and disadvantages, of quantum computation and a possible model for hybrid hierarchical quantum search. We will strive for presenting a accessible mathematical analysis of our hybrid approach which takes into account the referred objectives.

2 Branching factor ramifications

In theoretical computer science one possible way to measure a problem's complexity consists in assessing how long a given algorithm takes to find a solution. However, time performance is dependent on a multitude of hardware related factors. Accordingly, it is often more suitable to take appropriate steps to determine the total number of items that are to be evaluated. In the case of a classical tree search this equates to the number of nodes to take into account. From a classical tree search perspective complexity is expressed in terms of b , the branching factor or maximum number of successors of any node; d , the depth of the shallowest goal node; m , the maximum length of any path in the state space [Russell et al., 2003]. Section 2.1 focuses on the aspects surrounding a constant branching factor. The requirements for a non-constant branching factor are presented in Section 2.2. The impact of using a non-constant branching factor is presented in Section 2.3.

2.1 Constant branching factor

As previously stated, our proposal for a hybrid quantum tree search system only relied on a constant branching factor b . This was mostly due to simplification reasons. However, a constant branching factor requirement is not feasible when considering potential applications of search algorithms. Since, at its essence, our system can be perceived as evaluating a superposition of all possible paths up to a depth level d , it is pivotal to determine the impact of a non-constant branching factor in our approach. Lets proceed by examining a couple of examples in order to have a clear understanding of the search process.

Figure 1 illustrates a search tree where at any given node it is always possible to apply two actions (respectively labeled as a_0 and a_1), i.e. the branching factor for this search tree is 2. Recall from Section 1.3 that we need to encode these actions in a binary fashion. In order to do so we need to determine how many bits n do we require. For an even number of actions we can simply calculate the base-2 logarithm. However, we need to take into account that an odd number of actions might be required. In this case we need to map the value into the next largest integer, which can be done through the ceiling function, i.e. $n = \lceil \log_2 |a| \rceil$ bits, where $|a|$ denotes the cardinality of the action set. Notice that the complete range of values allowed with n bits might not be used, i.e. $|a| < 2^n$. From our point of view, it is the unitary operator's responsibility to validate whether a binary string is an admissible action.

In the case of the search tree illustrated in Figure 1 which possesses a branching factor $b = 2$ actions we need $\lceil \log_2 2 \rceil = 1$ bit. Accordingly, let value 0 denote a_0 and value 1 represent a_1 . The binary strings encoding the paths leading to each leaf nodes of the search tree illustrated in Figure 1 are presented in Table 2.

Path to node	Action at level 1	Action at level 2	Action at level 3
H	0	0	0
I	0	0	1
J	0	1	0
K	0	1	1
L	1	0	0
M	1	0	1
N	1	1	0
O	1	1	1

Table 2: Binary encoding for each possible path of the search tree illustrated in Figure 1

Suppose we wish to perform a search up to depth level d . We can easily build a string of d elements, one for each possible depth, with each element requiring a binary representation using n bits. In total, our binary string will employ $n \times d$ bits. We are also able to construct a quantum superposition $|\psi\rangle$ encompassing all the actions to be applied up to depth level d as illustrated by Expression 4. This superposition $|\psi\rangle$ can then be employed alongside our unitary operator U and Grover's algorithm.

$$|\psi\rangle = \frac{1}{\sqrt{2^{n \times d}}} \sum_{x=0}^{2^{n \times d}-1} |x\rangle \quad (4)$$

2.2 Non-constant branching factor

Now consider the tree presented in Figure 2 with an action set $a = \{a_0, a_1, a_2, a_3, a_4\}$. The first thing one notices is that we no longer have a constant branching factor at each node. In fact for this particular case it is convenient to distinguish between two types of branching factor, namely, the theoretical maximum branching factor $b_{max} = |a| = 5$, and the average branching factor of each node $b_{avg} = 2$, not including the leafs. In order to encode each of the possible actions we require $n = \lceil \log_2 |a| \rceil = 3$ bits. Let the encodings of each action be those presented in Table 3. Accordingly, the binary strings leading to each leaf node are illustrated in Table 4.

action	b_0	b_1	b_2
a_0	0	0	0
a_1	0	0	1
a_2	0	1	0
a_3	0	1	1
a_4	1	0	0
undefined	1	0	1
undefined	1	1	0
undefined	1	1	1

Table 3: Binary encoding for each possible path of the search tree illustrated in Figure 2

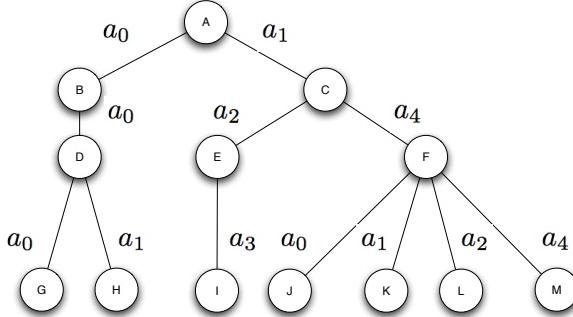


Fig. 2: A search tree with a maximum branching factor $b_{max} = 5$ and an average branching factor $b_{avg} = \frac{2+1+2+2+1+4}{6} = 2$.

Path to node	Action at level 1	Action at level 2	Action at level 3
G	000	000	000
H	000	000	001
I	001	010	011
J	001	100	000
K	001	100	001
L	001	100	010
M	001	100	100

Table 4: Binary encoding for each possible path of the search tree illustrated in Figure 1

In order to employ Grover's algorithm we would need to build a quantum superposition state $|\psi'\rangle$ similar to that presented in Expression 4. Again, our superposition would consist of those states with d elements, each of which with n bits. However, there is a crucial difference between both tree searches. Consider the case illustrated in Figure 2 where a search to depth level $d = 3$ alongside $b_{max} = 5$ is performed. Superposition $|\psi'\rangle$ would contain all the quantum states belonging to the range $[0, 2^{d \times n} - 1] = [0, 2^9 - 1] = [0, 511]$. I.e. we would be able encode 512 possible paths when in reality we would only need to encode the states presented in Table 4 **. In reality, the vast majority of the states present in the superposition would contain inadmissible configurations of actions.

It is important to draw attention to the fact that Grover's algorithm provides a quadratic speedup $O(\sqrt{N})$ where N is the number of elements present in the superposition. By employing $n = \lceil \log_2 b_{max} \rceil$ bits, when in practice $n = \lceil \log_2 b_{avg} \rceil$ bits might have sufficed, we are extending the search space and in the process loosing some of the speedup provided by Grover's algorithm.

Naturally, the question arises: Considering the above encoding mechanism, and a $b_{avg} < b_{max}$ when does Grover stop providing a speedup over classical approaches?

** An alternative approach would consist in encoding each possible state, instead of encoding in an binary fashion the sequence of actions. However, this would have a meaningful impact on the complexity and design of unitary operator U since each admissible input string would have to be mapped onto a predefined sequence of actions.

2.3 Analysis

So, how can we proceed in order to analyze the problem depicted in the previous section? As with so many other fields it is usually easier to start out with an example and extrapolate from that. Accordingly, lets consider the following scenario: we want to perform a tree search using our hybrid quantum search system up to depth level 10; the maximum branching factor, b_{max} is 5, however on average we are only able to perform three actions, i.e. $b_{avg} = 3$. Does our approach still provides an advantage over classical search strategies?

In order to answer this question we need to consider the complexities of classical search algorithms. Traditionally, these methods experience some type of exponential growth in the number of leaf nodes that need to be assessed. Since the number of elements to be evaluated is a function of the branching factor b and the depth of the search d the associated complexity is typically of the form $O(b^d)$. Clearly, in the case of our scenario we need to differentiate between the two branching factors, respectively b_{max} and b_{avg} . Accordingly, if we consider b_{max} then a total of $O(b_{max}^d) = O(5^{10}) = 9765625$ nodes might potentially need to be evaluated. On the other hand, by employing b_{avg} a total of $O(b_{avg}^d) = O(3^{10}) = 59049$ nodes may be considered. These values differ by a factor of ≈ 165 which is considerable when in practice it is acceptable to consider b_{avg} as the *de facto* branching factor.

Now lets reflect on the number of times one would need to apply Grover's iterate. In this case there is no distinction to be made between b_{max} and b_{avg} since we would still need to build a quantum superposition state encoding all of the b_{max} actions up to a depth level d . As previously stated this would result in a total of $n \times d = 3 \times 10 = 30$ bits being required, where $n = \lceil \log_2 b_{max} \rceil = \lceil \log_2 5 \rceil = 3$. Accordingly, for the above scenario we would need to apply Grover's iterate a total of $O(\sqrt{N}) = O(\sqrt{2^{30}}) = 32768$ times in order to obtain a solution. By comparing the number of states classically evaluated by using b_{avg} against the total number of iterations required by Grover's algorithm we see that both values differ by a factor of ≈ 1.8 . Not surprisingly, although we still obtain a speedup over the classical approach it is severely lessened. Intuitively, it should be clear that this behaviour can be perceived in the following manner:

- As b_{avg} grows closer to b_{max} the number of times to apply Grover's iterate will be optimal relatively to the classical approaches;
- As b_{avg} grows more distant to b_{max} the number of Grover iterations to apply will be closer to b_{avg}^d .

From the above reasoning we are now able to successfully determine the number of times that Grover's iterate should be applied, respectively $|G|$, as illustrated by Expression 5.

$$\begin{aligned}
|G| &= \sqrt{N} \\
&= \sqrt{2^{n \times d}} \\
&= \sqrt{2^{\lceil \log_2 b_{max} \rceil \times d}} \\
&= 2^{\frac{\lceil \log_2 b_{max} \rceil \times d}{2}}
\end{aligned} \tag{5}$$

Keep in mind that we wish to determine where the threshold lies between the number of elements of a classical search, respectively b_{avg}^d and the total number of times to apply Grover's

iterate $|G|$. This process can be formulated as presented in Expression 6 which when solved results in Expression 7.

$$b_{avg}^d = |G| \quad (6)$$

$$\Leftrightarrow b_{avg}^d = 2^{\frac{\lceil \log_2 b_{max} \rceil}{2} d} \quad (7)$$

Accordingly, when $b_{avg} < 2^{\frac{\lceil \log_2 b_{max} \rceil}{2}}$ then the total number of nodes evaluated in classical search will be less than the number of times to apply Grover's iterate, i.e. $b_{avg}^d < |G|$. Appropriately, when $b_{avg} > 2^{\frac{\lceil \log_2 b_{max} \rceil}{2}}$ then our hybrid system will yield a speedup over classical search algorithms. The plot of Expression 7, for $b_{max} \in [2, 128]$, is presented in Figure 3. The shaded area indicates those values of b_{avg} that will produce better performance results classically over the proposed hybrid search.

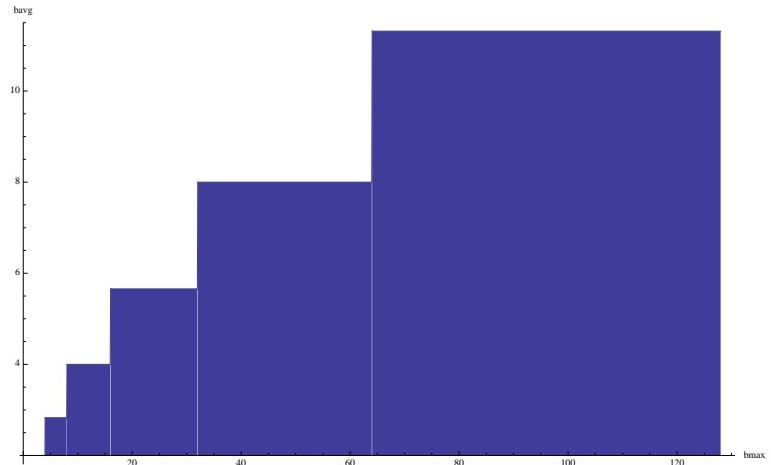


Fig. 3: The area plot of $b_{avg} \leq 2^{\frac{\lceil \log_2 b_{max} \rceil}{2}}$ for $b_{max} \in [2, 128]$. The shaded area indicates those values of b_{avg} that will produce better performance results over our hybrid quantum production system.

Figure 3 also showcases the characteristic ladder effect of functions employing the ceiling function required by the branching factor binary coding mechanism. Notice that in practice this means that there will always be ranges of values for b_{max} where the number of iterations, $|G|$, will remain the same. Consequently, the average branching factor, b_{avg} , for the boundary condition presented in Expression 6 will also remain constant in those intervals. This happens despite the fact that b_{max} is growing in the associated range.

Lets proceed by elaborating a bit more on each b_{max} and the associated range of b_{avg} values. In order to encode in a binary fashion any b_{max} value we require a total of $n = \lceil \log_2 b_{max} \rceil$ bits. The use of the ceiling function effectively forces certain ranges of b_{max} values to require

the same number of n bits. As we have seen previously, the number of Grover iterations is a function of the total number of bits required. We can see how this influences b_{avg} by taking Expression 7 into account. Clearly, for those b_{max} values requiring the same number of bits, the associated b_{avg} values will remain the same. It is only when the required number of bits for b_{max} changes that an impact will be felt regarding b_{avg} . Accordingly, we are interested in studying what happens when a transition if performed from, for example, n to $n + 1$ bits. In this case the b_{avg} value grows from $2^{\frac{n}{2}}$ to $2^{\frac{n+1}{2}}$ which differ by a factor of $\sqrt{2}$. Let b_{avg}^n denote the average branching factor b_{avg} associated with an interval requiring n bits. Then, it is possible to express b_{avg}^{n+1} as a function of b_{avg}^n . This recurrence behaviour is illustrated in Expression 8.

$$b_{avg}^{n+1} = \sqrt{2} b_{avg}^n \quad (8)$$

Expression 8 can be improved if we allow ourselves to leave behind the use of the ceiling function employed in Expression 7. In doing so, we are deliberately abdicating of the ladder effect presented in Figure 3 and giving the b_{avg} function a quadratic form. This new formulation is presented in Expression 9.

$$b_{avg} = 2^{\frac{\log_2 b_{max}}{2}} = \sqrt{b_{max}} \quad (9)$$

Since we no longer have a “range-bounded” function, a direct mapping of Expression 8 is impossible. Instead, we can simply define a function, b'_{avg} , depicting the new upper-range values of b_{avg} , as illustrated in Expression 10. This $\sqrt{2}$ -constant growth factor is depicted in Figure 4.

$$b'_{avg} = \sqrt{2} b_{avg} = \sqrt{2b_{max}} \quad (10)$$

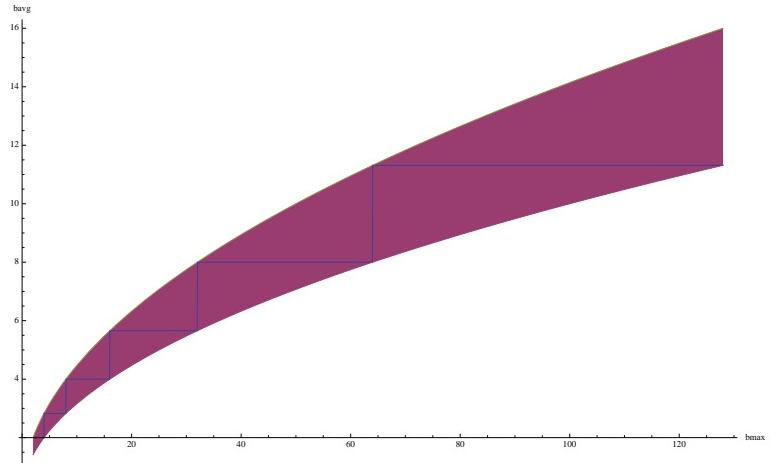


Fig. 4: The $\sqrt{2}$ -constant growth between b'_{avg} and b_{avg} superimposed on the original $b_{avg} = 2^{\frac{\log_2 b_{max}}{2}}$ function for $b_{max} \in [2, 128]$.

3 Heuristic Perspective

Determining which production rule should be applied to the working memory is an important part of the control strategy of production system theory. In a great deal of occasions an artificial intelligence application does not possess the adequate level of knowledge allowing for full differentiation amongst the production rules [Nilsson, 1982].

At any given point in time during the search process it would be useful to somehow know which production might produce a state which is closer to a goal state. Intuitively, we may describe this process as trying to determine the quality of a path of actions with an optimal solution having the lowest path cost among all solutions [Russell et al., 2003]. A key component of these systems, and many other algorithms in artificial intelligence, consists in an heuristic function $h(n)$ responsible for presenting an estimate of the distance that a given state n is relatively to a goal state. Function $h(n)$ is typically employed alongside a function $g(n)$ which reflects the search cost incurred to reach state n . Traditionally, the conjunction of both these functions is incorporated within a single evaluation function $f(n)$ as illustrated by Expression 11.

$$f(n) = g(n) + h(n) \quad (11)$$

Not surprisingly, function $h(n)$ can have a number of different definitions depending on the specifics of the problem at hand. In a sense, heuristic functions are responsible for providing extra-information about how-well a search is performing. Production systems need not depend on the use of heuristics. Systems which do not take into consideration any kind of auxiliary information are referred to as uninformed strategies. In the case of “uninformed” production systems the choice of which rule to apply is performed at random. On the other hand, “informed” strategies enable a production system to select an appropriate rule.

Our quantum production system model can be perceived as systematically trying to apply actions in an uninformed fashion. This operation is performed until a goal state is reached. Such behaviour resembles that of a standard blind depth-limited search process. Intuitively, the heuristic concepts incorporated into informed search strategies appear to be an adequate extension idea to our quantum production system. However, it remains to be seen if these concepts can be adequately mapped into our approach and if doing so provides an advantage. The remainder of this section is organized as follows: Section 3.1) considers how to incorporate the use of an heuristic the unitary operator; Section 3.2) provides an analysis of the quantum computation procedure incorporating the use of an heuristic. Section 3.3) builds on these results to present an extended quantum heuristic mechanism.

3.1 The Quantum Heuristic

As previously mentioned, at its core, our system can be perceived as evaluating a superposition of all possible paths. Expression 4 illustrated this perspective, where a quantum superposition state is composed by an amplitude value and a multitude of sub-states, i.e. the computational basis. From a quantum mechanics perspective it is important to reinforce the idea that the mechanisms allowing quantum states of a superposition to communicate with the other states are complex and restricted in what they achieve. Those algorithms that are able to provide a

meaningful speedup do so by employing, and determining, some type of global property. E.g. Grover's algorithm is able to determine the mean amplitude A of a quantum superposition and perform an inversion about the mean [Grover, 1996]. Also Shor's algorithm for factoring numbers is able to efficiently determine the period of periodic quantum states [Shor, 1994]. Accordingly, there is no clear method to communicate between states.

This fact immediately poses a problem: traditionally, the use of heuristics is employed to choose among possible tree paths, ideally producing an optimal sequence of actions. However, with our system, we are unable to quantum mechanically perform such a comparison. Ergo, is there any way to incorporate any heuristic concepts into our hybrid approach?

From a simplified point of view, an heuristic function outputs a value estimating the distance to a goal. We can therefore opt to consider only those states whose f value is below a certain threshold T . Notice that in doing so we are deliberately ignoring the fact that these same states may later on provide an optimal solution path. In order to for the search process to incorporate the heuristic function we need to reflect it upon the unitary operator's design. Recall that a unitary operator U in order to be employed by Grover's algorithm is only required to flip the amplitudes of the solutions states. Expression 12 reflects both of the previous requirements, where $|n\rangle$ represents the current node being processed and a_i an action taken at depth i .

$$U|b\rangle|a_1a_2\cdots a_d\rangle = \begin{cases} -|b\rangle|a_1a_2\cdots a_d\rangle & \text{iff } (b, a_1, a_2, \dots, a_d) \leq T \\ |b\rangle|a_1a_2\cdots a_d\rangle & \text{otherwise} \end{cases} \quad (12)$$

In a similar way as to deciding which of two possible heuristics to employ, the matter of deciding the threshold value T could be left to statistical studies, or even informed intuition based on hands-on experience [Nilsson, 1982]. Take notice that in no way did we circumvent the problem of comparing multiple paths. In conclusion, we are able to incorporate some of the concepts surrounding the use of heuristics, but not all of them.

3.2 Interpretation

What happens when we employ a unitary operator U with the form presented in Expression 12 alongside Grover's algorithm? I.e. do we stand to gain anything by incorporating heuristic concepts into our hybrid search system? Section 3.2 illustrates how superposition $|\psi\rangle$ can be decomposed into two components. Section 3.2) provides a graphical illustration of the impacts of performing search up to different depth levels.

Decomposing the superposition state Answering this question requires extending our analysis of what happens when unitary operator U is applied to superposition $|\psi\rangle$, i.e. $U|\psi\rangle$. Suppose superposition $|\psi\rangle$ takes the form illustrated in Expression 13, where n represents the number of bits.

$$|\psi\rangle = \sqrt{\frac{1}{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \quad (13)$$

Before advancing any further, notice that in an uniform superposition, associated with each state $|x\rangle$ there is an amplitude $\alpha \in \mathbb{C}$, which in this case is $\sqrt{\frac{1}{2^n}}$ for all states. Let α_i denote the amplitude associated with state $|i\rangle$. Quantum computation requires the norm of amplitudes to be unit-length, i.e. $\sum_{x=0}^{2^n-1} |\alpha_x|^2 = 1$, at all times. Not surprisingly any unitary operator must also preserve the norm.

Now note that we can decompose our initial quantum superposition $|\psi\rangle$ into two parts [Kaye et al., 2007]. One part will contain all those states that are solutions, which we will respectively label as set X_{good} . Another part will include the non-solutions states, respectively labeled as set X_{bad} . Also, assume that the problem we are interested in solving contains k solutions. This implies that $|X_{good}| = k$ and $|X_{bad}| = 2^n - k$. Accordingly, an uniform superposition of the states belonging to X_{good} would set an equal amplitude among its k elements, i.e. $\sqrt{\frac{1}{k}}$. By the same reasoning, an uniform superposition of the states in X_{bad} would impose an amplitude values of $\sqrt{\frac{1}{2^n-k}}$. Accordingly, we can define the superposition states $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$, respectively presented in Expression 14 and Expression 15.

$$|\psi_{good}\rangle = \sqrt{\frac{1}{k}} \sum_{x \in X_{good}} |x\rangle \quad (14)$$

$$|\psi_{bad}\rangle = \sqrt{\frac{1}{2^n - k}} \sum_{x \in X_{bad}} |x\rangle \quad (15)$$

We are now able to express superposition $|\psi\rangle$ in terms of the subspaces $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$. This process is illustrated in Expression 16 which is indispensable to the rest of our analysis.

$$|\psi\rangle = \sqrt{\frac{k}{2^n}} |\psi_{good}\rangle + \sqrt{\frac{2^n - k}{2^n}} |\psi_{bad}\rangle \quad (16)$$

The requirement that the norm must be preserved at all times induces a probabilistic behaviour. Consider state $|\psi\rangle$ whose norm is $\left| \sqrt{\frac{k}{2^n}} \right|^2 + \left| \sqrt{\frac{2^n - k}{2^n}} \right|^2 = 1$. I.e. we have a sum of values which sum up to 1 similarly to a probability distribution. Accordingly, the probability of obtaining state $|\psi_{good}\rangle = \left| \sqrt{\frac{k}{2^n}} \right|^2$ and state $|\psi_{bad}\rangle = \left| \sqrt{\frac{2^n - k}{2^n}} \right|^2$. Applying Grover's iterate effectively changes the amplitudes, maximizing the probability of obtaining a solution contained in the subspace spawned by $|\psi_{good}\rangle$. As a result, the amplitude associated with state $|\psi_{good}\rangle$ will increase. Since the norm of state $|\psi\rangle$ must be preserved, employing Grover also implies that the amplitude of $|\psi_{bad}\rangle$ will be decreased.

Heuristic Impact In order to continue with our analysis lets assume we possess an admissible heuristic which always eliminates candidate states with each additional level of depth. Although this is a rather optimistic strategy the heuristic's behaviour can be viewed as ideal. We will use this best case scenario to demonstrate the system's potential performance.

Given a sufficiently high depth level d the heuristic will have to eventually produce the exact number of solutions k . Let k_d denote the number of solutions at depth level d . Then, with such an heuristic we would have $k_1 < k_2 < \dots < k_d \leq k$. However, we can never produce fewer than k solutions, since that would violate basic assumptions about the search space of the problem.

The above process can be visualized geometrically. Notice that our initial superposition $|\psi\rangle$ containing k solutions can be understood as vector with two components $(|\psi_{good}\rangle, |\psi_{bad}\rangle) = (\sqrt{\frac{k}{2^n}}, \sqrt{\frac{2^n-k}{2^n}})$. Therefore, we are able to map it into a two dimensional plane with axis $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$. This mapping process is illustrated in Figure 5. From a quantum computation perspective, we can envisage the use of different states $|\psi_{k_d}\rangle$ reflecting a search up to depth level d . State $|\psi_{k_d}\rangle$ is a simple reformulation in terms of k_d of state $|\psi\rangle$, as presented in Expression 17.

$$|\psi_{k_d}\rangle = \sqrt{\frac{k_d}{2^n}} |\psi_{good}\rangle + \sqrt{\frac{2^n - k_d}{2^n}} |\psi_{bad}\rangle \quad (17)$$

Clearly, as the number of solutions k_d grows, to the allowable maximum of k , the $|\psi_{good}\rangle$ and $|\psi_{bad}\rangle$ components will tend towards the values of the original state vector $|\psi\rangle$. This behaviour is illustrated in Expression 18.

$$\lim_{k_d \rightarrow k} \left(\sqrt{\frac{k_d}{2^n}}, \sqrt{\frac{2^n - k_d}{2^n}} \right) = \left(\sqrt{\frac{k}{2^n}}, \sqrt{\frac{2^n - k}{2^n}} \right) \quad (18)$$

Not surprisingly, as the depth of the search increases the corresponding state vector $|\psi_{k_d}\rangle$ gets closer to the original $|\psi\rangle$, as can be perceived from Figure 5. Additionally, each state $|\psi_{k_d}\rangle$ can be understood as forming an angle θ whose tangent is shown in Expression 19.

$$\tan \theta_{k_d} = \frac{\sqrt{\frac{k_d}{2^n}}}{\sqrt{\frac{2^n - k_d}{2^n}}} = \sqrt{\frac{k_d}{2^n - k_d}} \quad (19)$$

Using this approach we can perform a comparison between the angles of different search depth levels. Lets say we wish to compare how the search advanced between depth levels d_1 and d_2 . We can define an operator $\Delta\theta_{k_{d_1}, k_{d_2}}$ with the behaviour defined in Expression 20.

$$\Delta\theta_{k_{d_1}, k_{d_2}} := \underbrace{\arctan \sqrt{\frac{k_{d_2}}{2^n - k_{d_2}}}}_{\theta_{k_{d_2}}} - \underbrace{\arctan \sqrt{\frac{k_{d_1}}{2^n - k_{d_1}}}}_{\theta_{k_{d_1}}} \quad (20)$$

Operator $\Delta\theta_{k_{d_1}, k_{d_2}}$ provides a mechanism for determining the operational success of adding $(d_2 - d_1)$ extra levels of depth relatively to d_1 . Accordingly, small $\Delta\theta_{k_{d_1}, k_{d_2}}$ values can be understood as not contributing in a significant manner to changing the system's overall state. Conversely, high $\Delta\theta_{k_{d_1}, k_{d_2}}$ values reveal that the system's state significantly shifted towards $|\psi\rangle$.

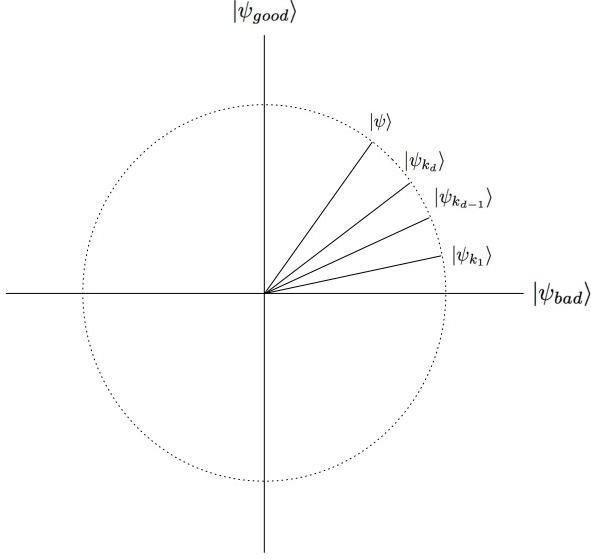


Fig. 5: The geometric interpretation of the original state vector $|\psi\rangle$ alongside different state vectors $|\psi_i\rangle$ reflecting the state attained by performing a search to depth-level i . Deeper searches will be closer to the original $|\psi\rangle$ state. All states are unit-length vectors.

Ultimately, we can only expect to get as far as state $|\psi\rangle$. From this point on Grover's usual $O(\sqrt{N})$ iterations will still be required in order to perform a rotation of $|\psi\rangle$ towards $|\psi_{good}\rangle$, leaving the algorithms overall complexity unchanged.

3.3 Extending the quantum heuristic

Can the concepts of the quantum heuristic be extended in such a way as to perform some kind of useful task? In order to answer this question we will assume that the heuristic function employed has some kind of probabilistic distribution. Accordingly, we will start by reviewing some principles surrounding heuristic distributions. We will then show how to incorporate these concepts alongside our hybrid quantum search proposal.

Heuristic distributions Lets assume that the heuristic function employed has a probabilistic distribution, i.e. the probability of each output value is between 0 and 1. This behaviour can be written as $0 \leq P(X = x) \leq 1$ where X is a random variable representing an output event, and x is a possible value of X . Additionally, a random variable may be either discrete or continuous depending on the values that it can assume. Random variables whose set of possible values belong to \mathbb{Z} are said to be discrete. On the other other hand, continuous variables map events to an uncountable set such as \mathbb{R} . For a discrete random variable X , the sum of the set containing all possible probability values is 1 as illustrated by Expression 21.

$$\sum_{i=1}^n P(X = x_i) = 1 \quad (21)$$

As a concrete example of a discrete heuristic we can consider the sliding block puzzle, also known as the n -puzzle, search problem. A sliding block puzzle challenges a player to shift pieces around on a board without lifting them to establish a certain end-configuration, as illustrated in Figure 6. This non-lifting property makes finding moves, and the paths opened up by each move important parts of solving sliding block puzzles [Hordern, 1987]. Accordingly, we can define a heuristic function h_1 which simply calculates the number of misplaced tiles between a board and a target board configuration. Function h_1 outputs values belonging to the range $[0, 9]$ and consequently can be classified as discrete. The discrete probability distribution for function h_1 regarding the 8-puzzle is presented in Figure 7.

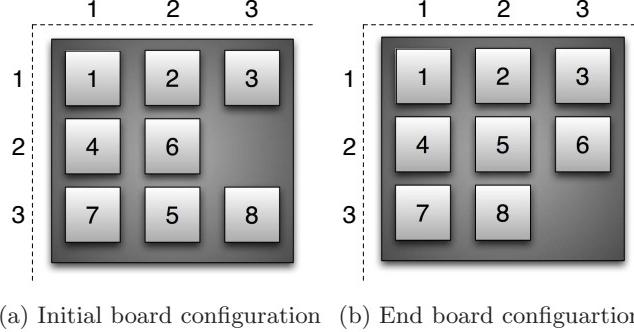


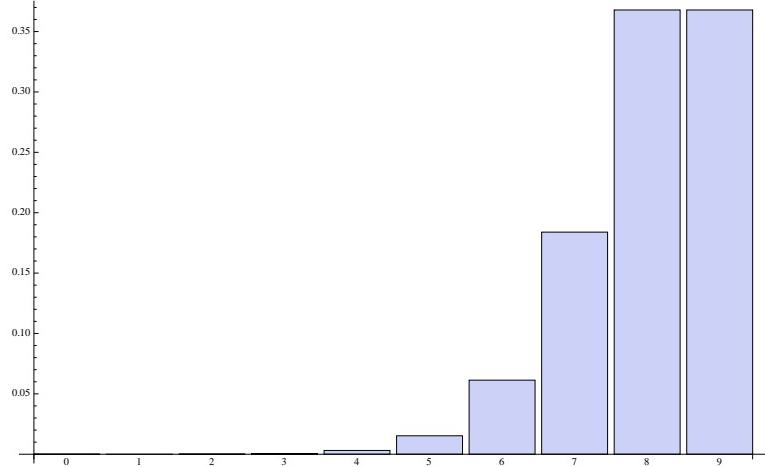
Fig. 6: A sliding block puzzle example with a board of dimension 3×3 , also known as the 8-puzzle.

If X is a continuous random variable then there exists a nonnegative function $P(X)$, the probability density function of the random variable X . The density function $P(X = c)$ is defined as the ratio of the probability that X falls into an interval around c , divided by the width of the interval, as the interval width goes to zero [DeGroot and Schervish, 2002]. This process is illustrated in Expression 22. Additionally, the density function must be nonnegative for all arguments and must obey the behaviour presented in Expression 23 [Russell et al., 2003].

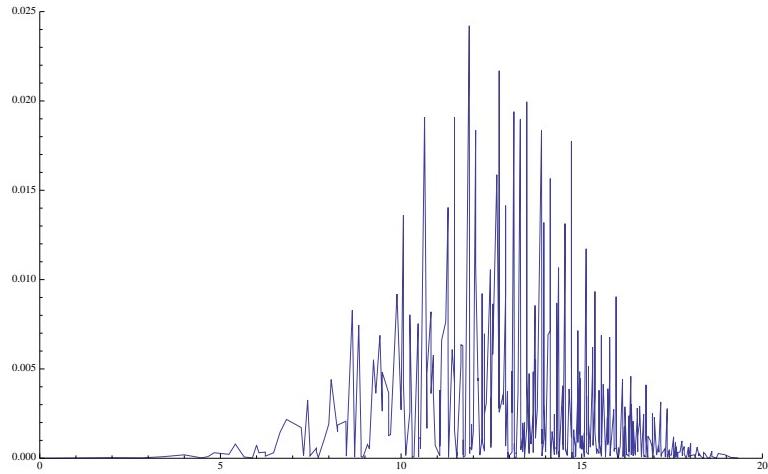
$$P(X = c) = \lim_{dx \rightarrow 0} P(c \leq X \leq c + dx)/dx \quad (22)$$

$$\int_{-\infty}^{+\infty} P(X)dx = 1 \quad (23)$$

As an example of a continuous heuristic function we can again consider the n -puzzle search problem. This time we need only to determine an heuristic function h_2 mapping values into a real codomain. This can be done by employing a metric such as the euclidean distance. For

Fig. 7: Discrete probability distribution for h_1 when applied to the 8-puzzle

instance, we can define h_2 in such a fashion as to calculate the euclidean distance for all the corresponding elements of a board and a target board configuration. I.e., $h_2 = \sum_{\forall \text{ tiles}} d(l_{\text{tile}}, l'_{\text{tile}})$, where d is the euclidean distance, l_{tile} the location of a tile in a board configuration and l'_{tile} the location of the corresponding tile in the target configuration. The probability density function for h_2 regarding the 8-puzzle is presented in Figure 8.

Fig. 8: Continuous probability density function for h_2 when applied to the 8-puzzle

The probability that a random variable X takes on a value that is less than or equal to x is referred to as the cumulative distributive function F which has the form shown in Expression 24.

$$F(x) = P(X \leq x) \quad (24)$$

The cumulative distribution function $F(a)$ for a discrete random variable is a simple sum of the values up to element a . This behaviour is illustrated in Expression 25. Similarly, a cumulative probability density function $F(a)$ can be defined for a continuous random variable as shown in Expression 26.

$$F(a) = \sum_{\text{all } x \leq a} P(X = x) \quad (25)$$

$$F(a) = \int_{-\infty}^a P(x)dx \quad (26)$$

Now suppose we wish to determine when, that is for which values, the cumulative distribution function equals a certain probability p , i.e. $F(x) = p$. In probability theory, this mapping is known as the quantile function of a cumulative distribution function $F(x)$ and is expressed as $F^{-1}(p) = x$. Once more we need to be careful in order to differentiate between discrete and continuous random variables. In the former, there may exist gaps between values in the domain of the cumulative distribution function, accordingly F^{-1} is defined as presented in Expression 27 where \inf denotes the infimum operator [Gilchrist, 2000].

$$F^{-1}(p) = \inf\{x \in \mathbb{R} : p \leq F(x)\} \quad (27)$$

In the case of a continuous random variable obtaining a clear expression for the quantile function is not so trivial since it requires determining the inverse of an integral. Notwithstanding, it is still possible to determine such expressions, for instance, the quantile function of a normal distribution with mean μ and standard deviation σ is illustrated in Expression 28, where erf is the Gauss error function [Gilchrist, 2000].

$$F^{-1}(p, \mu, \sigma^2) = \mu + \sqrt{2}\sigma erfc^{-1}(2p - 1), \quad p \in [0, 1] \quad (28)$$

Extended quantum heuristic Naturally, the question arises, how can this process be combined with our hybrid quantum search approach? It turns out that when a fourth of all possible states analyzed by Grover's algorithm are marked as solutions, i.e. $k = \frac{1}{4}N$ then a single iteration is required in order to obtain with certainty one of the solution states [Hirvensalo, 2004]. Accordingly, we can try to fine tune the behaviour of our unitary operator such that it outputs a solution for a fourth of all cases. This procedure can be performed with the assistance of the quantile function concept introduced in the previous section.

Lets say we have an heuristic function $f : X \rightarrow Y$ and are interested in obtaining the states which are closest to a solution. Accordingly, we are interested in marking as a solution those states that produce the smallest values of codomain Y . Lets assume that the states which are closer to a goal node are those who tend to happen less times (as exemplified by Figure 7 and Figure 8). From a probabilistic point-of-view it is possible to check if the heuristic value is

less than or equal to the quantile function output for a probability of 25%. This behaviour is illustrated in Expression 29.

$$U|b\rangle|a_1a_2\cdots a_d\rangle = \begin{cases} -|b\rangle|a_1a_2\cdots a_d\rangle & \text{iff } (b, a_1, a_2, \dots, a_d) \leq F^{-1}(0.25) \\ |b\rangle|a_1a_2\cdots a_d\rangle & \text{otherwise} \end{cases} \quad (29)$$

Ideally, by using this approach it is possible to obtain a superposition containing one fourth of the “closest” states to a goal configuration by applying a single iteration of Grover’s algorithm. Additionally, we can further expand on the results of Expression 29 in order to contemplate different sections of a probability distribution. For instance, we can choose to obtain in a single iteration of Grover’s algorithm the states which lie between heuristic values ($F^{-1}(0.5) - F^{-1}(0.25)$). Similarly, we could also choose to obtain the states belonging to ($F^{-1}(0.75) - F^{-1}(0.5)$) or ($F^{-1}(1) - F^{-1}(0.75)$). More generally, let a and b denote two probabilities values such that $b - a = 0.25$, then it is possible to determine if a given heuristic value belongs to the range $[F^{-1}(a), F^{-1}(b)]$. This strategy for selecting 25% of the search space is presented in the unitary operator shown in Expression 30,

$$U|b\rangle|a_1a_2\cdots a_d\rangle = \begin{cases} -|b\rangle|a_1a_2\cdots a_d\rangle & \text{iff } (b, a_1, a_2, \dots, a_d) \in [F^{-1}(a), F^{-1}(b)] \\ |b\rangle|a_1a_2\cdots a_d\rangle & \text{otherwise} \end{cases} \quad (30)$$

In a certain sense, employing this type of procedure allows for a kind of partial selection of the search space to be performed using a single Grover iteration. The selection is only partial because upon measurement a random collapse amongst the marked states is obtained.

4 Final considerations

It is important to mention that some of the graph dynamics considered in this work relate directly to the well-studied quantum random walks on graphs (for an introduction to this research area please refer to [Ambainis, 2003], [Kempe, 2003] and [Ambainis, 2004]. Quantum random walks are the quantum equivalents of their classical counterparts (we refer the reader to [Hughes, 1995] [Woess, 2000] for basic facts regarding random walks). Quantum random walks were initially approached in [Aharonov et al., 1993], [Meyer, 1996], [Nayak and Vishwanath, 2000] and [Ambainis et al., 2001] in one-dimensional terms, i.e. walk on a line. The system is described in terms of a position n on the line and a direction d , i.e. $|n\rangle|d\rangle$ in the Hilbert space $\mathcal{H} = \mathcal{H}_n \otimes \mathcal{H}_d$ where \mathcal{H}_n is the Hilbert space spanned by the basis vectors encoding the position and \mathcal{H}_d the Hilbert space spanned by the vectors of the direction. The direction register, sometimes referred to as the coin space, is initialized to a superposition of the possible direction, in the case of a walk on a line, either left or right, and the position n updated based on the direction of the walk. The choice of which superposition to apply is also a matter investigated. Surprisingly, if the system is executed for t steps it behaves rather differently than its classical random walks. Specifically, the authors found that the system spreads quadratically faster over the line than its classical equivalent.

Some of the first approaches proposing quantum random walking on graphs can be found in [Farhi and Gutmann, 1998], [Hogg, 1998], [Aharonov et al., 2001] and [Childs et al., 2002]. Let $G(V, E)$ represent a d -regular graph, and let \mathcal{H}_V be the Hilbert space spanned by states $|v\rangle$

where $v \in V$, and \mathcal{H}_E be an Hilbert space of dimension d spanned by basis states $|1\rangle, \dots, |d\rangle$. Overall, the system can be described through basis states $|v\rangle|e\rangle$ for all $v \in V$ and $e \in E$. The common approach is to randomly select one of the edges e adjacent to v and the update the current position of the graph, i.e. $U|v\rangle|e\rangle \rightarrow |v'\rangle|e\rangle$, if e has edge points v and v' . The random selection may also be performed using a d -dimensional coin space. Perhaps more interesting is the hitting time, i.e. the time it takes to reach a certain vertex B starting from a vertex A . In [Childs et al., 2002] and [Childs et al., 2003] a graph example is presented where a classical random walk would take $\Omega(2^d)$ steps to reach B , where d is the depth of the graph. However, the quantum equivalent walk can reach B in $O(d^2)$ computational steps, providing for an exponential speedup! Other examples of quantum random walks include how to adapt the models to perform a search [Shenvi et al., 2003], [Ambainis, 2004], [Ambainis et al., 2005], and [Ambainis, 2007]

Not surprisingly, due to the hard computational problems being asked, the vast majority of these approaches put a strong emphasis on actually determining if a node can be reached, and if so how fast. Consequently, questions about the path leading to a pre-specified node have not been properly addressed. For instance, this fact is explicitly pointed out in [Childs et al., 2003], namely: “Note that although our algorithm finds the name of the exit, it does not find a particular path from entrance to exit”. For many artificial intelligence applications the ability to answer this question is a crucial one as it provides the basis for powerful inference mechanisms capable of knowledge deduction. The ability to obtain the full path open measurement is a key feature of the production system proposed in this work. Additionally, quantum random walks incorporate previous knowledge of a graph in the form of the relationship $G(V, E)$. This kind of knowledge may not always be available from start, e.g. systems where new nodes are generated based on the information accessible to a system at any given point in time.

The quantum random walk approach differs drastically from the model presented in the previous sections which focused on detailing some of the key notions supporting an hybrid quantum search system. Such a system incorporated classical search concepts, expressed through unitary operators, with Grover’s quantum search algorithm. From a classical point of view, applying a search procedure can be understood as partitioning the search space into blocks. Each block is then examined in order to determine if a solution is present. However, such an approach did not in any way performed a kind of partition of the “quantum search space”. Instead, it executed a sequence of actions in order to determine if a goal state was reached. So the question naturally arises: Is there any procedure to perform a hierarchical quantum search based solely on decomposition of the quantum search space?

Grover and Radhakrishnan were some of the first to purpose a possible approach to this problem in [Grover and Radhakrishnan, 2004]. The main motivation of the work was the following: consider a quantum search space containing a single solution is divided into l -blocks of equal size, suppose that we wish to determine in which of the l -blocks the solution is, can this process be performed with fewer queries than the original Grover algorithm? In practice, this problem reduces to the one of determining the first m bits of the n bit computational basis containing the solution, with $m \leq n$. The authors proceed by analysing what happens when a variation of Grover’s iterate for amplitude amplification is applied to each block. They conclude that it is indeed easier to determine the initial m bits. However, as m grows closer to n the computational gains obtained disappear [Grover and Radhakrishnan, 2004]. Later, an optimization to this approach was proposed in [Korepin and Xu, 2007], albeit only marginally improving the lower bound on the number of oracle queries.

A number of authors have also focused on extending Grover's original search algorithm. Namely, in [Zalka, 1999b] the quantum search algorithm was shown to be optimal in the sense that it gives the maximal possible probability of finding a solution. Other examples of possible extensions to Grover's original work include returning a solution with certainty (please refer to [Zalka, 1999a], [Brassard et al., 2000], [Long, 2001] and [Hu, 2002]) and assessing the impact of having multiple solutions [Boyer et al., 1998]. Additionally, it should be mentioned that no quantum black-box search algorithm can solve the search problem by making fewer than $\Omega(\sqrt{N})$ iterations. This result was first shown in [Bennett et al., 1997] and later revised on [Boyer et al., 1998].

5 Conclusions

In this work we examined the ramifications of an hybrid quantum search system. We chose to focus on two key aspects: the use of a non-constant branching factor and the adoption of an heuristic point of view. Clearly, both of these cases are not without its flaws. However, we consider the additional insight provided to be valuable. In the case of the non-constant branching factor we were able to verify that deciding whether or not to Grover, or to proceed classically, should take into account the maximum and the average branching factor. Additionally, by evaluating the states that are within a given threshold we are able to incorporate some classical heuristic concepts into our approach. These concepts were further extended with the use of probabilistic distribution functions allowing for a selection mechanism to be obtained. This mechanism enables specific ranges of quantum states to be obtained in an efficient manner.

6 Acknowledgements

This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds and FCT grant DFRH - SFRH/BD/61846/2009.

References

- Aharonov et al., 2001. Aharonov, D., Ambainis, A., Kempe, J., and Vazirani, U. (2001). Quantum walks on graphs. In *Proceedings of ACM Symposium on Theory of Computation (STOC'01)*, pages 50–59.
- Aharonov et al., 1993. Aharonov, Y., Davidovich, L., and Zagury, N. (1993). Quantum random walks. *Phys. Rev. A*, 48(2):1687–1690.
- Ambainis, 2003. Ambainis, A. (2003). Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1:507.
- Ambainis, 2004. Ambainis, A. (2004). Quantum search algorithms. *SIGACT News*, 35(2):22–35.
- Ambainis, 2007. Ambainis, A. (2007). Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37:210.
- Ambainis et al., 2001. Ambainis, A., Bach, E., Nayak, A., Vishwanath, A., and Watrous, J. (2001). One-dimensional quantum walks. In *ACM Symposium on Theory of Computing*, pages 37–49.
- Ambainis et al., 2005. Ambainis, A., Kempe, J., and Rivosh, A. (2005). Coins make quantum walks faster.

- Anderson, 1983. Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, Massachusetts, USA.
- Bennett et al., 1997. Bennett, C. H., Bernstein, E., Brassard, G., and Vazirani, U. (1997). Strengths and Weaknesses of Quantum Computing. *eprint arXiv:quant-ph/9701001*.
- Boyer et al., 1998. Boyer, M., Brassard, G., Hoyer, P., and Tapp, A. (1998). Tight bounds on quantum searching. *Fortschritte der Physik*, 46:493.
- Brassard et al., 2000. Brassard, G., Hoyer, P., Mosca, M., and Tapp, A. (2000). Quantum Amplitude Amplification and Estimation. *eprint arXiv:quant-ph/0005055*.
- Campbell et al., 2002. Campbell, M., Hoane Jr., A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial Intelligence*, 134:57–83.
- Childs et al., 2003. Childs, A. M., Cleve, R., Deotto, E., Farhi, E., Gutmann, S., and Spielman, D. (2003). Exponential algorithmic speedup by quantum walk. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC 2003)*, pages 59–68.
- Childs et al., 2002. Childs, A. M., Farhi, E., and Gutmann, S. (2002). An example of the difference between quantum and classical random walks. *Quantum Information Processing*, 1(1):35–43.
- Chuang et al., 1998. Chuang, I. L., Gershenfeld, N., and Kubinec, M. (1998). Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, 80(15):3408–3411.
- DeGroot and Schervish, 2002. DeGroot, M. H. and Schervish, M. J. (2002). *Probability and statistics*. Addison-Wesley, 3 edition.
- Dirac, 1939. Dirac, P. A. M. (1939). A new notation for quantum mechanics. In *Proceedings of the Cambridge Philosophical Society*, volume 35, pages 416–418.
- Dirac, 1981. Dirac, P. A. M. (1981). *The Principles of Quantum Mechanics - Volume 27 of International series of monographs on physics (Oxford, England) Oxford science publications*. Oxford University Press.
- Edmonds, 1965. Edmonds, J. (1965). Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467.
- Ernst and Newell, 1969. Ernst, G. and Newell, A. (1969). *GPS: a case study in generality and problem solving*. Academic Press, New York, NY, USA.
- Farhi and Gutmann, 1998. Farhi, E. and Gutmann, S. (1998). Quantum computation and decision trees. *Phys. Rev. A*, 58(2):915–928.
- Feldmann, 1993. Feldmann, R. (1993). *Game Tree Search on Massively Parallel Systems*. PhD thesis, University of Paderborn.
- Garey and Johnson, 1979. Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman.
- Gilchrist, 2000. Gilchrist, W. (2000). *Statistical Modelling with Quantile Functions*. Chapman and Hall/CRC.
- Grover, 1996. Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, New York, NY, USA. ACM.
- Grover, 1998a. Grover, L. K. (1998a). A framework for fast quantum mechanical algorithms. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 53–62, New York, NY, USA. ACM.
- Grover, 1998b. Grover, L. K. (1998b). Quantum computers can search rapidly by using almost any transformation. *Phys. Rev. Lett.*, 80(19):4329–4332.
- Grover, 1999. Grover, L. K. (1999). Quantum search on structured problems. *Chaos, Solitons & Fractals*, 10(10):1695 – 1705.
- Grover, 2002. Grover, L. K. (2002). Trade-offs in the quantum search algorithm. *Phys. Rev. A*, 66(5):052314.
- Grover, 2005. Grover, L. K. (2005). Fixed-point quantum search. *Phys. Rev. Lett.*, 95(15):150501.
- Grover and Radhakrishnan, 2004. Grover, L. K. and Radhakrishnan, J. (2004). Is partial quantum search of a database any easier? *eprint arXiv:quant-ph/0407122*.
- Hart et al., 1968. Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107.

- Hirvensalo, 2004. Hirvensalo, M. (2004). *Quantum Computing*. Springer-Verlag, Berlin Heidelberg.
- Hogg, 1998. Hogg, T. (1998). A framework for structured quantum search. *PHYSICA D*, 120:102.
- Hopcroft and Tarjan, 1973. Hopcroft, J. and Tarjan, R. (1973). Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378.
- Hordern, 1987. Hordern, E. (1987). *Sliding Piece Puzzles*. Recreations in Mathematics, No 4. Oxford University Press, USA.
- Hsu, 1999. Hsu, F.-h. (1999). Ibm's deep blue chess grandmaster chips. *Micro, IEEE*, 19(2):70–82.
- Hsu, 2002. Hsu, F.-h. (2002). *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton University Press.
- Hu, 2002. Hu, C.-R. (2002). A family of sure-success quantum algorithms for solving a generalized grover search problem.
- Hughes, 1995. Hughes, B. (1995). *Random Walks and Random Environments: Volume 1: Random Walks*. Random Walks and Random Environments. Oxford University Press, USA.
- Kaye et al., 2007. Kaye, P. R., Laflamme, R., and Mosca, M. (2007). *An Introduction to Quantum Computing*. Oxford University Press, USA.
- Kempe, 2003. Kempe, J. (2003). Quantum random walks - an introductory overview. *Contemporary Physics*, 44:307.
- Korepin and Xu, 2007. Korepin, V. E. and Xu, Y. (2007). Hierarchical Quantum Search. *International Journal of Modern Physics B*, 21:5187–5205.
- Korf, 1985. Korf, R. E. (1985). Depth-first iterative-deepening : An optimal admissible tree search. *Artificial Intelligence*, 27(1):97 – 109.
- Korf, 1991. Korf, R. E. (1991). Best-first search with limited memory. *UCLA Computer Science Annual*.
- Korf, 1993. Korf, R. E. (1993). Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78.
- Laird et al., 1987. Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.
- Laird et al., 1986. Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in soar: The anatomy of a general learning mechanism. *Machine Learning*, 1(1):11–46.
- Long, 2001. Long, G. L. (2001). Grover algorithm with zero theoretical failure rate. *Phys. Rev. A*, 64(2):022307.
- Luger and Stubblefield, 1993. Luger, G. F. and Stubblefield, W. A. (1993). *Artificial Intelligence: Structures and Strategies for Complex Problem Solving: Second Edition*. The Benjamin/Cummings Publishing Company, Inc, Menlo Park, CA, USA.
- Mano and Kime, 2002. Mano, M. and Kime, C. R. (2002). *Logic and Computer Design Fundamentals: 2nd Edition*. Prentice Hall, Englewood Cliffs, NJ, USA.
- Meyer, 1996. Meyer, D. (1996). From quantum cellular automata to quantum lattice gases. *Journal of Statistical Physics*, 85(5):551–574.
- Moore, 1959. Moore, E. (1959). The shortest path through a maze. In *Proceeding of an International Symposium on the Theory of Switching, Part II*, pages 285–292, Cambridge, Massachusetts. Harvard University Press.
- Nayak and Vishwanath, 2000. Nayak, A. and Vishwanath, A. (2000). Quantum walk on the line. Technical report, DIMACS Technical Report.
- Newell, 1963. Newell, A. (1963). A guide to the general problem-solver program gps-2-2. Technical Report RM-3337-PR, RAND Corporation, Santa Monica, CA, USA.
- Newell and G, 1965. Newell, A. and G, E. (1965). The search for generality. In *Information Processing 1965: Proceeding of IFIP Congress*, volume 1, pages 17–24, Chicago. Spartan.
- Newell et al., 1959. Newell, A., Shaw, J., and Simon, H. A. (1959). Report on a general problem-solving program. In *Proceedings of the International Conference on Information Processing*, pages 256–264.
- Nielsen and Chuang, 2000. Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, MA, USA.
- Nilsson, 1982. Nilsson, N. J. (1982). *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc.

- Post, 1941. Post, E. (1941). *The Two-Valued Iterative Systems of Mathematical Logic. (AM-5)*. Princeton University Press.
- Post, 1943. Post, E. (1943). Formal reductions of the general combinatorial problem. *American Journal of Mathematics*, 65:197–268.
- Russell et al., 2003. Russell, S. J., Norvig, P., Canny, J. F., Edwards, D. D., Malik, J. M., and Thrun, S. (2003). *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.
- Santos et al., 2009a. Santos, A. C., Tarrataca, L., and João, C. (2009a). An analysis of navigation algorithms for smartphones using j2me. In *In Proceedings of the Second International ICST Conference on MOBILE Wireless MiddleWARE, Operating Systems, and Applications (Mobilware'09, Berlin-Germany, April 28-29), LNICST*, volume 7, pages 266–279. Springer.
- Santos et al., 2009b. Santos, A. C., Tarrataca, L., and João, C. (2009b). Context inference for mobile applications in the upcase project. In *In Proceedings of the Second International ICST Conference on MOBILE Wireless MiddleWARE, Operating Systems, and Applications (Mobilware'09, Berlin-Germany, April 28-29), LNICST*, volume 7, pages 352–365. Springer.
- Shenvi et al., 2003. Shenvi, N., Kempe, J., and Whaley, K. B. (2003). Quantum random-walk search algorithm. *Phys. Rev. A*, 67(5):052307.
- Shor, 1994. Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.
- Slate and Atkin, 1977. Slate, D. and Atkin, L. R. (1977). Chess 4.5 - northwestern university chess program. In *Chess Skill in Man and Machine*, pages 82–118. Berlin. Springer-Verlag.
- Toffoli, 1980a. Toffoli, T. (1980a). Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644, London, UK. Springer-Verlag.
- Toffoli, 1980b. Toffoli, T. (1980b). Reversible computing. Technical report, Massachusetts Institute of Technology, Laboratory for Computer Science, Massachusetts, MA, USA.
- Woess, 2000. Woess, W. (2000). *Random Walks on Infinite Graphs and Groups*. Number 138 in Cambridge Tracts in Mathematics. Cambridge University Press.
- Zalka, 1999a. Zalka, C. (1999a). A grover-based quantum search of optimal order for an unknown number of marked elements.
- Zalka, 1999b. Zalka, C. (1999b). Grover's quantum searching algorithm is optimal. *Physical Review A*, 60:2746.